# CS 378 (Spring 2003)

# Linux Kernel Programming

**Yongguang Zhang**

**(ygz@cs.utexas.edu)**

Copyright 2003, Yongguang Zhang

# This Lecture

- Linux Networking
  - Netfilter
  - Route cache and route lookup
  - Neighbour and link
  - Continue with output chain
  - Input chain
- Questions?

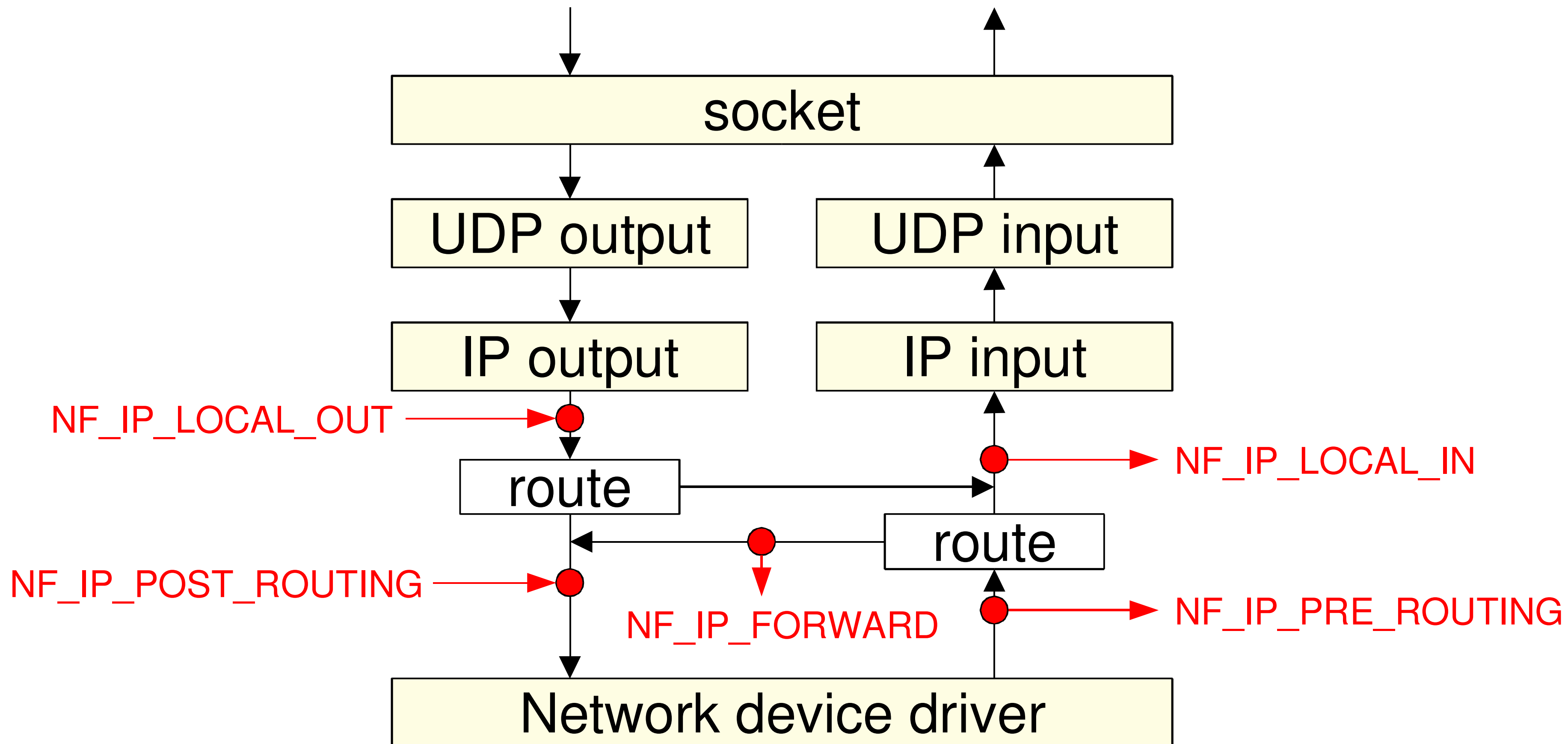

© 2003 Yongguang Zhang

# Netfilter

- A framework for packet mangling in kernel
- Hooks
  - The linux network stack provides “hooks” in its input/output chains
  - Kernel function (which wants to mangle the packets) registers to be called at certain hooks
  - When a packet reaches a hook, it is passed to the registered callback function
  - When function returns, the packet continues the chain
- Each protocol family can have different hooks
  - IPv4: 5 hooks


© 2003 Yongguang Zhang

# Netfilter Hooks (IPv4)




 © 2003 Yongguang Zhang 

# Example Use

- 5 functions registered at NF_IP_LOCAL_IN hook
  - ipt_route_hook(), ipt_hook(), ip_nat_fn(), ip_confirm(), fw_confirm()
- Example:
  - User space firewall command to stop SYN flood:

    ```
    iptables -t filter -A INPUT -p tcp --sync -m limit --limit 10/s
    ```
- You can add new functions to any hook
  - Write a callback hook function

    ```
    unsign int nf_hookfn( hooknum, struct sk_buff **pskb,
       in_dev, out_dev, int ( *okfn)(struct sk_buff *) );
    ```
  - Register it


© 2003 Yongguang Zhang

# Hook Callback Function

- Format: must be of function type nf_hookfn

```
unsign int nf_hookfn( hooknum, struct sk_buff **pskb,
    in_dev, out_dev, int ( *okfn)(struct sk_buff *) );
```

  – hooknum: the function call is triggered at this hook
  – pskb: a reference pointer to the packet (sk_buff)
  – in_dev, out_dev: the net device this packet is from/to
  – okfn: currently not used
- Hook function returns a "verdict"
  – NF_ACCEPT: should continue at next hook function
  – NF_DROP: kernel should drop the packet
  – NF_QUEUE, NF_STOLEN, NF_REPEAT

 © 2003 Yongguang Zhang 

# Register a Hook Callback Function

- Fill in a netfilter hook operation structure
  - Data type: (in include/linux/netfilter.h)

```
struct nf_hook_ops {
        struct list_head list;
        nf_hookfn *hook;            /* the callback function */
        int pf;                     /* the network family */
        int hooknum;                /* the hook number */
        int priority;               /* which hook goes first */
};
```

- Call this function to register at the hook
  - int nf_register_hook(struct nf_hook_ops *reg);

 © 2003 Yongguang Zhang 

# Hook Function Organization

- Kernel maintains a list of all hooks
  - 2-dim array nf_hooks (defined in net/core/netfilter.c)

    struct list_head nf_hooks[NPROP][NF_MAX_HOOKS]
  - 1st subscript for network family, 2nd for hook number
  - Each element is a list of all functions at the same hook, sorted by their priority
- Function nf_register_hook():
  - Insert the record at the proper list at the proper place
- Function nf_iterate():
  - Call all functions at the given hook list one-by-one
  - Return early if a verdict is not NF_ACCEPT


© 2003 Yongguang Zhang

# Invoking Netfilter Hook

- Linux kernel calls NF_HOOK macro:
  - NF_HOOK(pf, hook, skb, indev, outdev, okfn)
  - What it does: if hook empty (no hook functions) call okfn(skb), otherwise call nf_hook_slow()
- nf_hook_slow() invokes all hook functions
  - Call nf_interate() to call all functions one-by-one
  - If return verdict is NF_ACCEPT, call okfn(skb)
  - If return verdict is NF_DROP, free the packet (calling kfree_skb(skb)) and return


© 2003 Yongguang Zhang

# Route Table and Destination Cache

- Ultimate routing table in Linux kernel: FIB (Forwarding Information Base)
  - Can be big and complex (e.g., $10^3$ entries in "BFR")
  - To inspect the entire route table: /proc/net/route
- Recently used route entry (one per destination)
  - Stored in a slab cache (called route/destination cache)
  - Access via route cache hash table (rt_hash_table[])
  - Cache information/decisions about this destination
  - Different from FIB's route table entry (multiple destinations may share same entry in FIB, e.g. Subnet)
  - To inspect route cache: /proc/net/rt_cache

 © 2003 Yongguang Zhang 

# Route Cache Entry

- Data type: struct rtable
  - Defined in include/net/route.h
- Include a destination entry as the first part

```
union {
        struct dst_entry dst;
        struct rtable *rt_next;
} u;
```

  - rt_next used to link hash collision list
  - rt_next overloads with first element (a pointer) in dst
- Other major fields
  - Cache lookup key: struct rt_key key


© 2003 Yongguang Zhang

# Destination Entry

- Data type: struct dst_entry
  - Defined in include/net/dst.h
- Major fields:
  - First field: struct dst_entry *next (to overload with the rt_next pointer in struct rtable)
  - Network device: struct net_device  *dev;
  - Path characteristics: pmtu, rtt, ssthresh, cwnd, ...
  - The next-hop neighbor: struct neighbour *neihghour
  - Cached link-layer header: struct hh_cache *hh;
  - Input function: int (*input)(struct sk_buff*);
  - Output function: int (*output)(struct sk_buff*);


© 2003 Yongguang Zhang

# Route Lookup

- ip_route_output() and ip_route_output_key()
  - For route lookup in output chain
  - Find the matching route in route cache
  - If not, call ip_route_output_slow()
- ip_route_output_slow()
  - Look up the route from FIB
  - Build a route cache entry (e.g., assign output() action)
- ip_route_input() and ip_route_input_slow()
  - For use in input chain, slightly different from output


© 2003 Yongguang Zhang

# Route Action

- The input, output function in route cache entry
  - Actions for this destination
- Possible output() functions:
  - ip_output(), ip_mc_output(), ip_rt_bug()
- Possible input() functions:
  - ip_local_deliver(), ip_mr_input(), ip_forward(), …
- Assigned when the route cache is created
  - Example, in ip_route_output_slow():

```
rth->u.dst.output = ip_output;
if ....
        rth->u.dst.output = ip_mc_output;
```


© 2003 Yongguang Zhang

# Neighbor and Link

- Dealing with the data link layer, next-hop neighbors, and address translation (ARP)
- Recently seen neighbors are cached
  - Including hardware (link-layer) header (e.g., the Etherent header)
- Two major data structures for each neighbor:
  - Neighbor data table (struct neighbour) and link-layer header cache (struct hh_cache)
  - Each destination cache entry has pointers to these two
- Source code: net/core/neighbour.c


© 2003 Yongguang Zhang

# Brief Neighbor/Link Data Structures

- struct hh_cache (in include/linux/netdevice.h)
  – The output function: hh_output(skb)
  – Content of the cached header: hh_data[]
  – In most cases, hh_output() → dev_queue_xmit()
- struct neighbour (in include/net/neighbour.h)
  – The output function: output(skb)
  – Information and states about the address resolution
  – Packet queue waiting for ARP: arp_queue
  – Depending on actual ARP protocol, output() is usually neigh_resolve_output()


© 2003 Yongguang Zhang

# Continue the Output Chain

- We traced this far last time (assuming UDP/IP)
  - sys_sendto()
  - sock_sendmsg()
  - inet_sendmsg()
  - udp_sendmsg()
  - ip_build_xmit()
- More about destination cache
  - udp_sendmsg() calls
    - ip_route_output(&rt, daddr, ufh.saddr, tos, ipc.oif);
  - ip_build_xmit() saves rt->u.dst in skb->dst

# Last Steps in IP Output

- Last step in ip_build_xmit()
  - To enter the NF_IP_LOCAL_OUT hook

    ```
    err = NF_HOOK(PF_INET, NF_IP_LOCAL_OUT, skb,
      NULL, rt->u.dst.dev, output_maybe_reroute);
    ```
  - When exit the hook, call output_may_reroute()
- output_maybe_reroute(skb)
  - Call route action skb->dst->output(skb)
  - For unicast destination, output function is ip_output()


© 2003 Yongguang Zhang

# IP Output (cont'd)

- ip_output()
  - Calls ip_finish_output()
- ip_finish_output()
  - To enter the NF_IP_POST_ROUTING hook

    NF_HOOK(PF_INET, NF_IP_POST_ROUTING, skb, NULL, dev, ip_finish_output2)
  - When exit the hook, call ip_finish_output2()
- ip_finish_output2()
  - If link-layer header is cached, call hh->hh_output()
  - Otherwise, do ARP through dst->neighbor->output()
  - For most case, both evenutally call dev_queue_xmit()

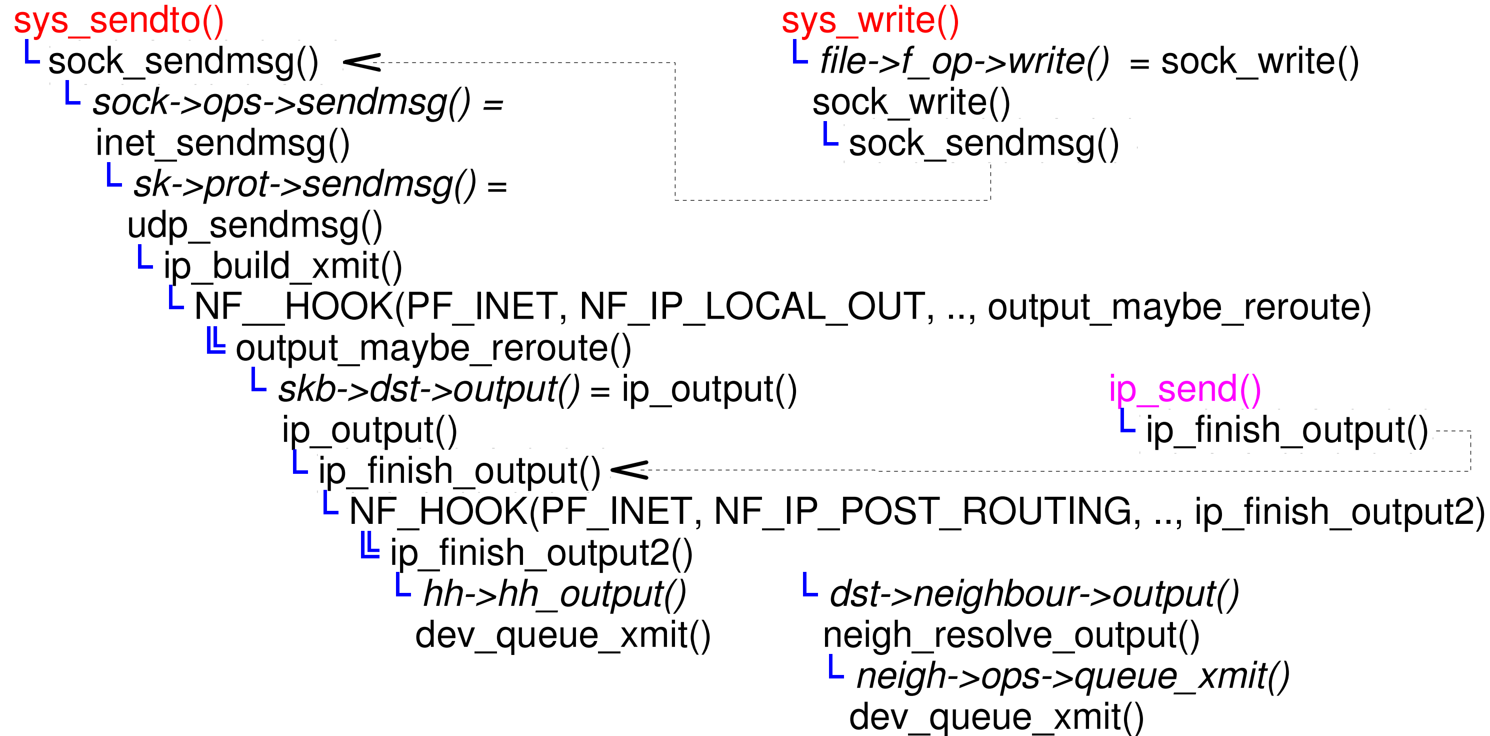

© 2003 Yongguang Zhang

# Queueing for Transmission

- dev_queue_xmit(skb)
  - If device has a queue discipline, enqueue the packet
    - q->enqueue(skb, q);
    - qdisc_run(dev); // kick the device queue!
  - If the device allows queuing the packet, send it
    - dev->hard_start_xmit()
  - All other cases: drop the packet
    - kfree_skb(skb)


© 2003 Yongguang Zhang

# Output Chain Function Calls





© 2003 Yongguang Zhang

# Tracing the Input Chain

- First trace from socket layer down, then from the device up
  - We will meet at the joint b/w socket and transport
  - Assume INET family and UDP protocol
- Start from recvfrom() system call (userspace)
  - System call service routine sys_recvfrom()
  - Which calls sock_recvmsg()
  - Which calls sock->ops->recvmsg()
    - PF_INET: sock->ops->recvmsg() → inet_recvmsg()
  - inet_recvmsg() calls sk->prot->recvmsg()
    - UDP: sk->prot->recvmsg() → udp_recvmsg()


© 2003 Yongguang Zhang

# UDP Receive Message

- udp_recvmsg()
  - Calls skb = skb_recv_datagram()
  - This may block during waiting
  - When it returns, copy the datagram to user space
  - Free skb (which is allocated down at the net device)
- skb_recv_datagram() essentially:
  - Dequeue packets from socket receive_queue

```
do {
        skb = skb_dequeue(&sk->receive_queue);
} while (wait_for_packet(sk, err, &timeo) == 0);
```

  - Who put packet in the queue? Lower layer.


© 2003 Yongguang Zhang

# Blocking Receive at Socket Layer

- skb_recv_datagram() essentially:
  - Dequeue packets from socket's receive_queue

```
do {
        skb = skb_dequeue(&sk->receive_queue);
} while (wait_for_packet(sk, err, &timeo) == 0);
```

- wait_for_packet()
  - Put this process into the sk->sleep wait queue, and give up CPU (set to TASK_INTERRUPTIBLE)
  - Upon return: remove me from sk->sleep and set process to TASK_RUNNING
- Who puts packets and wake the wait queue?


© 2003 Yongguang Zhang

# Input Chain from the Bottom

- Upon packet arrival, the network stack should
  - Put the packet onto sk->receive_queue
  - Wake up processes in sk->sleep queue
- Starting point: interrupt handling in a network device driver
  - Pull the packet from hardware
  - Allocate a socket buffer struct sk_buff
  - Calls netif_rx(skb)


© 2003 Yongguang Zhang

# Example from 3Com 3C501 Driver

- See drivers/net/3c501.c
- Interrupt handler: el_interrupt(irq, ...)
  - Check everything.
  - If it is a good receive packet, call el_receive(dev)
- el_receive(dev) essentially:

```
pkt_len = inw(RX_LOW);
skb = dev_alloc_skb(pkt_len+2);
...
    skb_reserve(skb,2);          /* force 16 byte alignment */
    skb->dev = dev;
    insb(DATAPORT, skb_put(skb,pkt_len), pkt_len);
    skb->protocol=eth_type_trans(skb,dev);
    netif_rx(skb);
```

# netif_rx()

- int netif_rx(struct sk_buff *skb)
  - Per CPU input packet queue (waiting for process): softnet_data[this_cpu].input_pkt_queue
  - Check for packet backlogs. Drop this packet if input packet queue is too long (too many backlogs).
  - Add skb to this CPU's input packet queue
  - Raise a soft irq (NET_RX_SOFTIRQ) on this CPU
- Still remember soft irq?
  - Run the tasklet right at the end of outer-most interrupt, or by softirqd


© 2003 Yongguang Zhang

# Network Receive Soft IRQ

- NET_RX_SOFTIRQ handler: net_rx_action()
  - Process all backlog incoming packets
- For each packet in this CPU's input packet queue
  - Remove from the input packet queue
  - Find the matching packet type (pt)
  - Call the corresponding input function for this type:
    - pt->func(skb, skb->dev, pt)
  - For INET family (with Ethernet packet type is IP), the corresponding input function is ip_rcv()


© 2003 Yongguang Zhang

# Network Layer Demultiplexing

- Kernel maintains a list of all network layer packet types we can handle
  - IP, IPX, IPv6, …
  - Data type: struct packet_type
    - unsigned short type;
    - int (*func) (struct sk_buff *, struct net_device *, ...);
- Each network family registers such a packet type
  - Example: ip_init()
    - ip_packet_type.type → ETH_P_IP
    - ip_packet_type.func → ip_rcv()
    - dev_add_pack(&ip_packet_type)

# IP Input

- ip_rcv()
  - Check the IP header, checksums, ...
  - Then go through the PRE_ROUTING netfilter hook

    NF_HOOK(PF_INET, NF_IP_PRE_ROUTING, skb, dev, NULL, ip_rcv_finish)
- ip_rcv_finish()
  - Call ip_route_input() to find the route for this packet
  - Result: a destination cache at skb->dst
  - skb->dst->input(): where it should go, possibly
    - ip_local_deliver(), ip_mr_input(), ip_forward(), …
  - Call skb->dst->input(skb)


© 2003 Yongguang Zhang

# The IP Forward Path

- For forwarding to other host: ip_forward()
  - Some checking, decrease TTL, ...
  - Then go through the FORWARD netfilter hook

    NF_HOOK(PF_INET, NF_IP_FORWARD, skb, skb->dev, dev2, ip_forward_finish)
- ip_forward_finish()
  - Calls ip_send() to inject the packet to the output chain


© 2003 Yongguang Zhang

# Protocol-Layer Demultiplexing

- For this host: ip_local_deliver(skb)
  - Does defragmentation if needed
  - Go through the LOCAL_IN netfilter hook

    NF_HOOK(PF_INET, NF_IP_LOCAL_IN, skb, skb->dev, NULL, ip_local_deliver_finish)
- ip_local_deliver_finish()
  - Find the matching protocol type from inet_protos[ ];
  - Calls ipprot->handler(skb)
  - For UDP: ipprot->handler() is udp_rcv()


© 2003 Yongguang Zhang

# UDP Receive

- udp_rcv()
  - Look up the corresponding socket structure (by UDP source, destination, ports)
    - sk = udp_v4_lookup(...);
  - Calls udp_queue_rcv_skb(sk,skb)
- udp_queue_rcv_skb(sk,skb)
  - Add packet to the socket, calling
    - sock_queue_rcv_skb(sk,skb)
  - Drop the packet if socket receive queue is full
    - kfree_skb(skb)

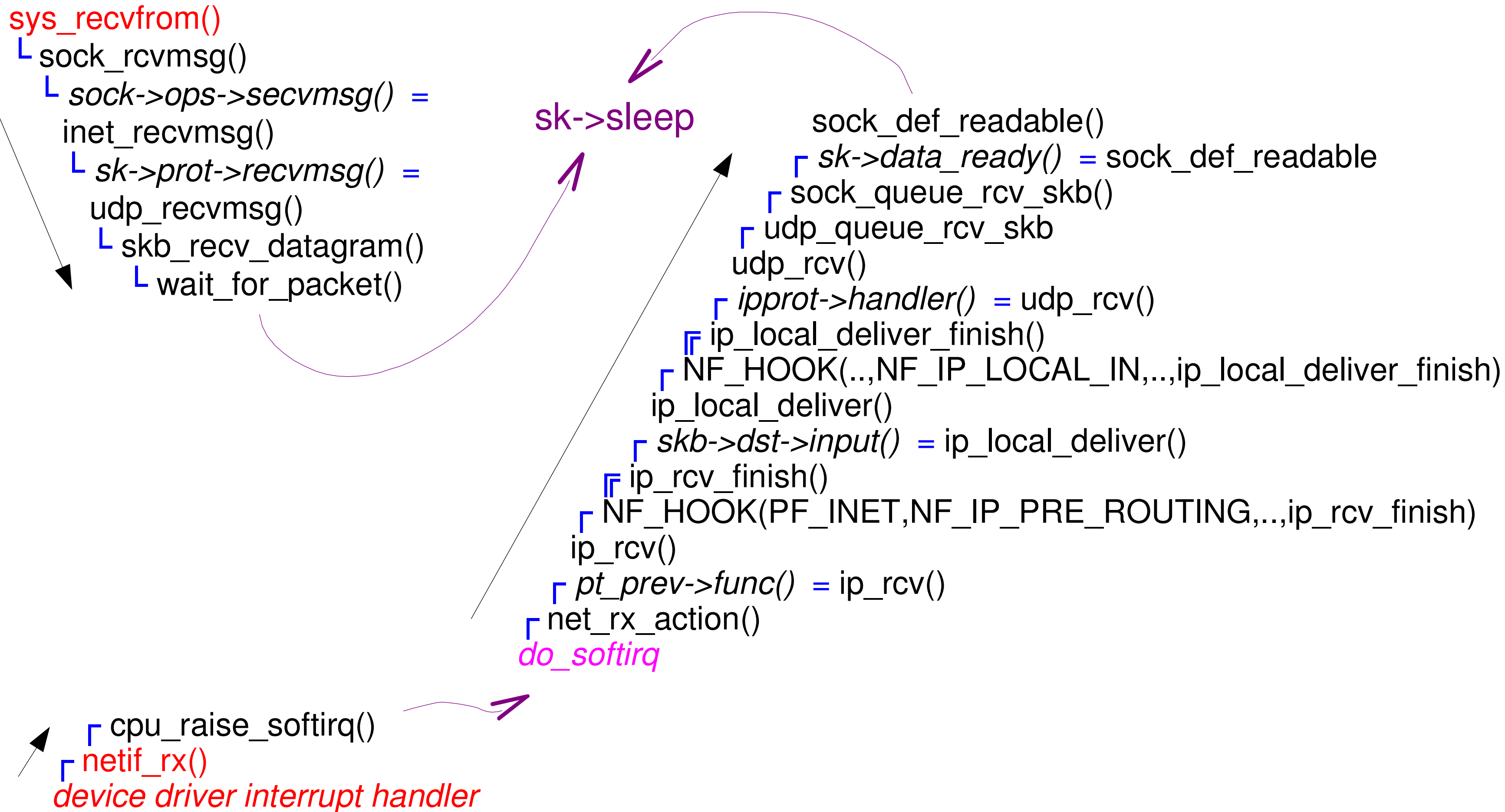

© 2003 Yongguang Zhang

# Meeting the Socket Layer

- Function sock_queue_rcv_skb(sk,skb)
  - Add the skb to the socket's receive queue
    - skb_queue_tail(&sk->receive_queue, skb)
  - Call the socket method that the data is ready
    - sk->data_ready(sk,skb->len)
- sk->data_ready()
  - The default function is sock_def_readable()
  - INET does not change this default
- sock_def_readable()
  - Wake up process in sk->sleep queue
    - wake_up_interruptible(sk->sleep);


© 2003 Yongguang Zhang

# Input Chain Function Calls




 © 2003 Yongguang Zhang 

# Adding a New Protocol

- For socket interface
  - Provide the set of protocol-specific socket methods (in struct proto type)
  - Add a record to the protocol switch (e.g., inetsw[] for INET)
- For demultiplexing (incoming)
  - Provide a receive handler function
  - Add a record to the protocol list (e.g., inet_protos[] for INET)


© 2003 Yongguang Zhang

# Summary

- Linux Networking
  - LKP: §8
  - ULK: §18
  - LDD2: §14 (network device driver)
- More about netfilter:
  - URL: http://www.netfilter.org/documentation/
- Next lecture:
  - Network Device Driver
  - Queueing Disciplines


© 2003 Yongguang Zhang